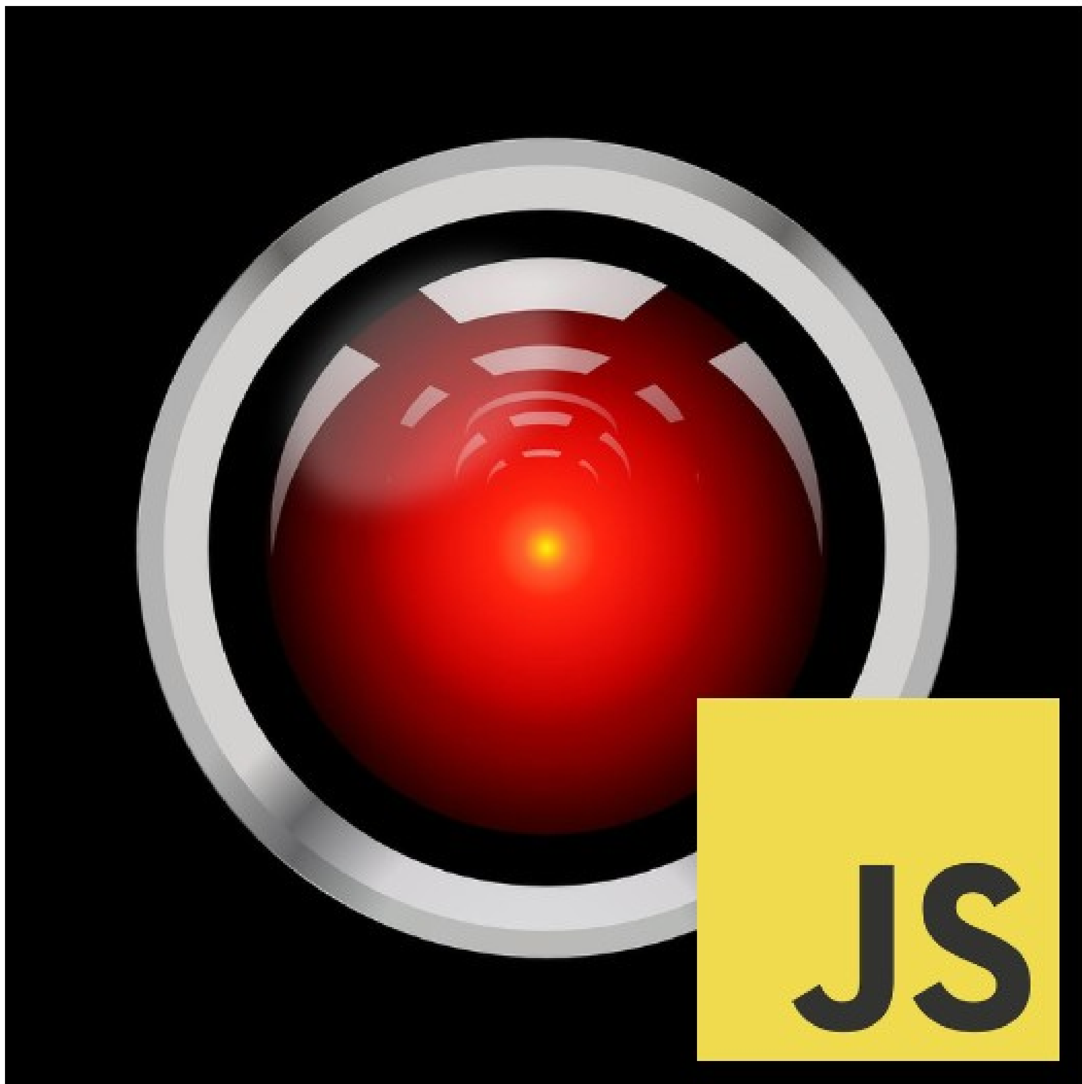
Принципи Нейронних мереж прямого поширення для JavaScript розробників

Автор:
Дмитро Попов
(linkedin.com/in/dima1popov/).

Дата:
2024.



Зміст

Коротка історія вивчення штучного інтелекту

Німецький математик Готфрід Лейбніц був першим, хто чітко висловив необхідність створення “універсальної характеристики”, тобто точної формальної мови, з допомогою якої можна обчислювати логічні розсуди, робити беззаперечні висновки. Перший, хто зробив впевнений крок в цьому напрямку, був англієць Джордж Буль. Буль запропонував декілька якісно нових ідей та підходів, які привели до створення булевої алгебри, логіки предикатів (Г.Фреге), теорії множин. Твердження, як операції над множинами, вивчали й до Буля, наприклад, Леонард Ейлер розробив Діаграми Ейлера, але Джордж Буль це все зібрав в єдину цілісну теорію. Парадокси наївної теорії множин були усунені в праці Бертрана Рассела та Альфреда Вайтгеда, там же був описаний синтаксис логіки предикатів на основі синтаксису італійця Джузеппе Пеано. Це все проклало шлях до створення “Символьного штучного інтелекту” (Symbolic AI), тобто машини, яка робить висновки на основі чітко прописаних правил логіки, типу правила Де Моргана, правила логіки Арістотеля, modus ponens та інші. Щоправда, Курт Гьодель показав, що тільки деякі висновки будуть несуперечливі, якщо користуватися методами Рассела та Вайтеда (та методами Давида Гілберта).
Алан Тюрінг після того, як дав формальний опис поняттю “обчислення” (на основі машини Тюрінга), зацікавився тим, як можна визначити чи має машина інтелект, і як наслідок запропонував Тест Тюрінга. Алан Тюрінг у статті "Обчислювальна техніка та інтелект" (1950) згадує важливість обмежень, які накладає Теорема Гьоделя про неповноту. Відповідно потрібно було знайти інші шляхи для створення мислячих машин. Програмісти почали надихатися біологією. У 1943 році нейрофізіолог Воррен Маккаллох і логік Волтер Піттс опублікували статтю, де змоделювали біологічну роботу органічного нейрона за допомогою електричних ланцюгів. Френк Розенблат розробив нейронну мережу прямого поширення. Потім штучний інтелект й розпізнавання образів вивчали Марвін Мінский та Джон Маккарті (розробник мови Lisp).
Вчений та письменник Айзек Азімов запропонував базову етику штучного інтелекту, а саме Три закони робототехніки.

Закони робототехніки Айзека Азімова:
1. Робот (ШІ) не може нашкодити людині або через свою бездіяльність допустити, щоб людині було завдано шкоди.
2. Робот мусить підкорятися наказам людини, коли ці накази не суперечать Першому закону.
3. Робот повинен дбати про свою безпеку доти, поки це не суперечить Першому і Другому законам.

Наприкінці 1980-х років праця Джуди Перла "Імовірнісне міркування в інтелектуальних системах" утвердили баєсові мережі як область дослідження.

В 1997 році машина Deep Blue здобула перемогу над чемпіоном світу з шахів, гросмейстером Гаррі Каспаровим. Цей матч вважається важливим етапом в історії шахів і штучного інтелекту, оскільки це був перший раз, коли світовий шаховий чемпіон програв комп'ютеру в матчі з класичним контролем часу.

Перші згадки про згорткові нейронні мережі (Convolutional Neural Networks) належать до робіт Яна ЛеКуна та його колег (див. Неокогнітрон Фокусіми). Одна з найбільш відомих робіт ЛеКуна та його команди описує LeNet-5, нейронну мережу, яка була успішно використана для розпізнавання рукописних цифр на базі даних MNIST.
База даних MNIST (скорочення від Mixed National Institute of Standards and Technology)

містить 60000 зображень для навчання і 10000 зображень для тестування.

У 2012 році Алекс Крижевський, Ілля Суцкевер та Джеффрі Гінтон представили архітектуру AlexNet, яка значно покращила результати у змаганнях з розпізнавання образів і стала поштовхом до популяризації глибокого навчання (Deep learning).

2013 року команда в Google під проводом Томаша Міколова створила word2vec, інструментарій вкладання слів, що може тренувати векторні моделі швидше за попередні підходи.

У 2014 році Ян Ґудфелоу та його колеги розробили генеративні змагальні мережі (GANs), які відкрили нові можливості для створення штучних зображень та інших даних.
Генеративна змагальна мережа (Generative Adversarial Network, GAN) — це клас алгоритмів машинного навчання, який складається з двох нейронних мереж, що змагаються одна з одною в процесі навчання. Ці дві мережі називаються генератором і дискримінатором.
Генератор покращує свої навички створення підроблених зразків, щоб дискримінатор не зміг їх відрізнити від реальних.
Дискримінатор покращує свої навички класифікації, щоб краще розрізняти реальні зразки від підроблених.
В 2017 році колектив DeepMind випустив препринт програми AlphaZero, яка після тренування протягом 24 годин змогла перемогти чемпіонів світу серед програм за іграми в шахи та го.
У 2017 році дослідники Google представили модель Transformer, яка революціонізувала обробку природної мови. Transformer використовує механізм уваги, що дозволяє моделі ефективніше обробляти великі послідовності даних. Це призвело до створення моделей BERT, GPT, та інших сучасних мовних моделей.
У 2020 році компанія OpenAI представила GPT-3 (Generative Pre-trained Transformer 3), одну з найбільш потужних мовних моделей на той час, яка здатна генерувати тексти, що майже не відрізняються від написаних людьми.

Визначення штучного інтелекту

> "Суть теоретичної моделі полягає в тому, що це система з відомими властивостями, які легко піддаються аналізу, яка, за гіпотезою, втілює істотні риси системи з невідомими або неоднозначними властивостями"
> (Френк Розенблат, "Персептрони й теорія мозку")

Багатошарові нейронні мережі вивчаються в рамках Глибокого навчання (Deep learning), Глибоке навчання є підмножиною Машинного навчання. Машинне навчання є підмножиною сфери Штучного інтелекту (ШІ).

Навчання — процес набуття знань та досвіду.

Контрольоване навчання (навчання з учителем, supervised learning) — це процес, коли модель навчається на позначеному наборі даних, тобто кожен навчальний приклад поєднується з вихідною міткою (правильною вдповіддю).

Очевидно, що людина з нароження має здатність *навчатися без учителя*. Мозок людини підсвідомо класифікує отриману інформацію. Інстинкти спонукають людину до навчання й дослідження, а рефлекси, біль та задоволення, працюють для зворотного зв'язку та підкріплення (reinforcement learning).
Нобелівський лауреат Іван Павлов писав: "Рефлекс — це реакція організму на зовнішній подразник".
У 1951 році голландський біолог Ніколас Тінберген опублікував "Дослідження інстинкту", впливову книгу про поведінку тварин. Ніколас Тінберген (1907 – 1988) отримав Нобелівську премію з фізіології та медицини 1973 року разом з Карлом фон Фрішем і Конрадом Лоренцем за їхні відкриття щодо організації та виявлення індивідуальних і соціальних моделей поведінки у тварин. Інстинкти слід відрізняти від рефлексів, які є простими реакціями організму на певний подразник, наприклад, звуження зіниць, як реакція на яскраве світло, колінний рефлекс, лоскіт, реакція "бийся або тікай". Можна сказати, що інстинкт є внутрішнім потягом до певної дії, якому не передують зовнішні подразники.

Методи машинного навчання включають кероване навчання (регресія, класифікація), некероване навчання (кластеризація (k-mean, DBSCAN), зменшення вимірності), напівкероване навчання, навчання з підкріпленням (Q-навчання, методи на основі політик), ансамблеві методи (беггінг, бустинг, випадковий ліс), глибоке навчання (згорткові нейронні мережі, рекурентні нейронні мережі, генеративно-змагальні мережі), машинне навчання на основі графів (графові нейронні мережі).

Підхід зверху вниз (Top-down Approach (Symbolic Reasoning)) моделює спосіб, яким людина розмірковує для розв'язування задач, використовуючи наявні знання та логічні правила. Цей підхід передбачає декомпозицію проблеми на підзадачі, які послідовно вирішуються, ґрунтуючись на загальних принципах і логічних правилах.
Підхід знизу вгору (Bottom-up Approach (Neural Networks)) моделює структуру людського мозку, що складається з великої кількості простих одиниць, званих нейронами. Кожен нейрон діє як зважене середнє своїх входів, і ми можемо навчити нейронну мережу вирішувати корисні задачі, надаючи їй навчальні дані.



Зверніть увагу, що сфера (множина) штучного інтелекту ширша ніж сфера машинного начання, а це означає, що машина може проявляти ознаки інтелекту, без вміння навчатися.

Штучний інтелект (Artificial Intelligence, AI) - це система, що демонструє ознаки інтелекту, а саме: вміння аналізувати, обчислювати, класифікувати, генерувати та зберігати інформацію.

“**Штучний інтелект** — здатність інженерної системи здобувати, обробляти, створювати та застосовувати знання та навички” (ISO/IEC TR 29119-11).

Зверніть увагу, що проходження відомого Тесту Тюрінга не є необхідною умовою, для того, щоб називати програму штучним інтелектом. В принципі, штучний інтелект підпадає під визначення, яке дав фізик Мічіо Кайку в книзі “Майбутнє розуму”: “створення моделі світу шляхом використання циклів зворотного зв’язку за різними параметрами (наприклад, за температурою, простором, часом), щоб досягти мети (наприклад, знайти товаришів, їжу, притулок)”.

Якщо ви не згодні з цим визначенням Штучного інтелекту, прочитайте про Ефект ШІ.

Тест Тюрінга (імітаційна гра): “Суддя взаємодіє з одним комп'ютером і однією людиною. На підставі відповідей на питання суддя повинен визначити, з ким він розмовляє: з людиною чи з комп'ютерною програмою. Завдання комп'ютерної програми — ввести суддю в оману, змусивши зробити неправильний вибір”. У червні 2012 року віртуальний співрозмовник Євген Густман (Eugene Goostman) виграв змагання на честь 100-річчя Алана Тюрінга, зумівши переконати 29% суддів, що він

людина.

“Інформація — це інформація, а не матерія чи енергія” (Норберт Вінер, “Кібернетика”)

DIKW піраміда



Дані: Сирі факти та цифри без контексту.
Інформація: Оброблені дані зі значенням.
Знання: Інформація, проаналізована для розуміння.
Мудрість: Застосування знань з інсайтом і судженням.

Як писав Конфуцій в “Лунь Юй”: Мудрий той, хто знає правильні шляхи.
Еразм Роттердамський у своїй книзі “Похвала глупоті” писав, що мудрий знає як чинити на благо.

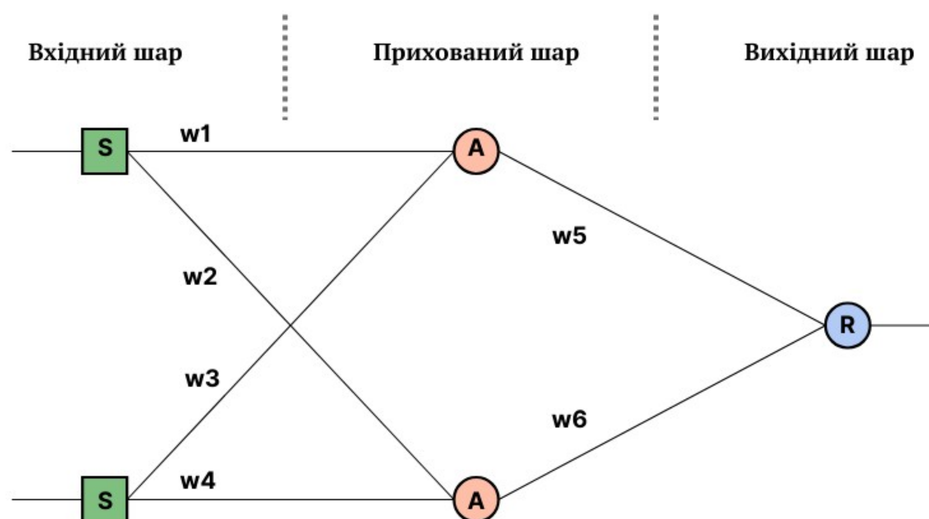
Принципи нейронних мереж прямого поширення

Тип нейронної мережі визначає загальну категорію або клас нейронних мереж, наприклад FNN, RNN, CNN.
Кожен тип нейронної мережі може мати певну архітектуру, наприклад, для RNN існує архітектура з довгою короткочасною пам'яттю (Long Short-Term Memory, LSTM), та Нейронна мережа Гопфілда.

Модель нейронної мережі є конкретною реалізацією архітектури з визначеними параметрами, які були навчені на певному наборі даних для виконання конкретного завдання. У вільному доступі є різні моделі штучного інтелекту (ШІ), зокрема Велика мовна модель (LLM, Large language model) під назвою Meta Llama 3.
FNN підходить для класифікації, RNN (LSTM) для обробки послідовностей, які подаються на вхід і мають залежності, CNN підходить для розпізнавання образів (OCR - Optical character recognition), Трансформери підходять для обробки природної мови (NLP). Для обробки природної мови машинами, текст ділиться на токени, потім токени перетворюються в числове (векторне) представлення, так званні ембеддінги. На основі цих ембеддінгів проводиться навчання моделі. В популярних бібліотеках, як PyTorch та Keras, є методи для цього (токенізації). За своєю суттю **токен** є найменшою одиницею, на яку можна розбити текстові дані для обробки в моделі ШІ.

Нейронні мережі прямого поширення (Feedforward neural network, FNN) є історично найпершими та найпростішими нейронними мережами. Вперше вони були описані Френком Розенблатом під назвою Персептрон в книзі 1962 року. Нейронна мережа прямого поширення (FNN) не має циклів, на відміну від рекурентних нейронних мереж (Recurrent Neural Networks, RNN), тобто мереж з циклами.
Нейронні мережі прямого поширення можна використовувати для різних задач, зокрема, для опису логічних операторів типу XOR, для розподілу листів на спам та не спам, для створення фільтрів картинок, для різного типу передбачень і рекомендацій, типу рекомендації фільмів, та щоб підібрати колір тексту відповідно до фону.
Френк Розенблат також довів, що багатошаровий Персептрон (FNN) може реалізувати оператор XOR та будь-які задачі класифікації відносно своїх входів.
З допомогою FNN (Feedforward neural network) можна навіть розпізнавати літери, проте для комплексних задач з розпізнавання образів, краще підходять Згорткові нейронні мережі (CNN, Convolutional neural network), які запропонував Ян Лекун.

Нейронну мережу прямого поширення (FNN) ми зобразимо у вигляді графа. Вузли (вершини) графа зображають штучні нейрони, по аналогії з природними нейронами у мозку. Нейрон приймає вхідні сигнали, сумує їх, та активується відповідно до функції активації, тобто подає сигнал на вихід. *Ми не враховуємо довжини зв'язків (ребер) на графі, і вважаємо, що сигнали розповсюджуються за однаковий час по всіх шляхах.*



Діаграма зображає Нейронну мережу прямого поширення (Feedforward neural network, FNN), яка здатна навчатися, щоб апроксимувати бінарні булеві оператори, типу XOR.

Уявимо, що кожен зв'язок w_i має певний електричний опір, отже, сигнал, який по ньому пройде, матиме силу відповідно до цього опору. Суть цієї нейронної мережі в тому, що ці опори (або ваги) динамічно змінюються під час навчання цієї мережі. Ми подаємо сигнали на вхідний шар (елементи S) й отримуємо результат на вихідному шарі (елемент R). Якщо результат нас влаштовує, ми підсилюємо ваги, якщо ні, зменшуємо ваги конкретних зв'язків. Шляхом такого зворотного зв'язку та підкріплення, мережа тренується. Перед навчанням ваги (опір у фізичних термінах) можна задати випадковим чином. Якщо мережа має сотні вузлів (нейронів), програміст не знає розподіл ваг після навчання, точніше він може їх переглянути, кудись зберегти, але в принципі йому це не потрібно знати, хіба він хоче налаштувати ще одну мережу подібним чином. Отже, конфігурація FNN мережі — це її структура та матриця ваг.
Штучний нейрон має функцію активації, тобто він передає сигнал на вихід, якщо сума його входів, пропущена через функцію активації, активує його. Деякі вузли можуть не мати функцію активації, наприклад вхідні елементи (S на діаграмі).

Для *функції активації* використовуються в різних випадках:

Розривна функція Хевісайда = x≥0 ? 1 : 0;

Сигмоїда (sigmoid, σ) = $e^x/(e^x+1) = 1/(1+e^{-x})$,

ReLU = max(0, x),
та інші.

Більшу перевагу надають “гладким функціям”, а не розривним, бо гладкі функції неперервно-

диференційовні, що дає можливість використовувати алгоритм градієнтного спуску для виправлення неточностей в результатах.

Похідна сигмоїди
$\sigma'(x) = \sigma(x) * (1 — \sigma(x))$;

Усунемо когнітивний дисонанс і перейдемо від опису нейронної мережі як фізичної структури з дротами, опором, електрикою та транзисторами, до опису *нейронної мережі як набору функцій та матриць.*
Нейрона мережа в математичних термінах, це набір функцій. Кожен шар мережі — це функція, яка приймає масив аргументів та повертає масив аргументів для наступного шару. Термін масив ми потім замінимо на вектор. Кожна така функція шару містить в собі функції, які представляють окремі нейрони (та функції їх активації) і матрицю ваг їх зв'язків. В ООП об'єкти шарів створюються з допомогою класів.

Матриця — таблиця чисел. Матриця з одним рядком, або з одним стовпцем називається вектором.
Для того, щоб можна було перемножити дві матриці A і B, кількість стовпців матриці A повинна дорівнювати кількості рядків матриці B, або навпаки.

Множення матриці можна визначити на основі *скалярного добутку векторів*.

a = <x,y>;
b = <v,w>;
a * **b** = xv + yw; (скалярний добуток).

Множення матриць

$$\vec{b_1}\ \ \vec{b_2}$$

$$\begin{matrix}\vec{a_1}\rightarrow\\ \vec{a_2}\rightarrow\end{matrix}\quad \begin{bmatrix}1 & 7\\ 2 & 4\end{bmatrix}\cdot\begin{bmatrix}3 & 3\\ 5 & 2\end{bmatrix}=\begin{bmatrix}\vec{a_1}\cdot\vec{b_1} & \vec{a_1}\cdot\vec{b_2}\\ \vec{a_2}\cdot\vec{b_1} & \vec{a_2}\cdot\vec{b_2}\end{bmatrix}$$

$$A \qquad\qquad B \qquad\qquad\qquad C$$

Тепер проведемо обчислення мережі, яка зображена вище на діаграмі.

Вхіднний шар

A =

1	0

B =

w_1	w_2
w_3	w_4

A * **B** =

1 * w_1 + 0 * w_3	1 * w_2 + 0 * w_4

Прихований шар

C =

sigmoid(1 * w_1 + 0 * w_3)	sigmoid(1 * w_2 + 0 * w_4)

C *

w_5
w_6

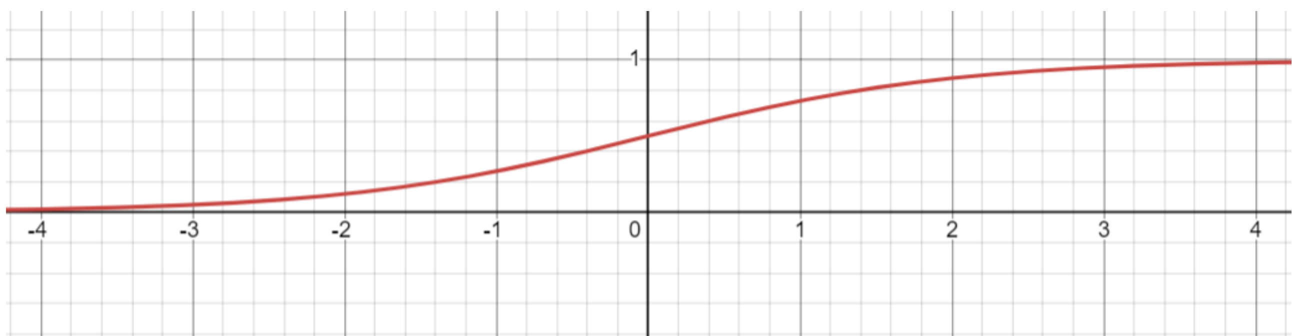
= sigmoid(sigmoid(1 * w_1 + 0 * w_3) * w_5 + sigmoid(1 * w_2 + 0 * w_4) * w_6).

Сумуємо ці значення й пропускаємо через *функію активації* (сигмоїду — логістичну функцію).

Bias (зсув) у feedforward neural networks (FNN) — це додатковий параметр, який додається до зважених сум вхідних значень перед їх передачею через активаційну функцію нейронів у кожному шарі мережі. Додавання bias дозволяє активаціям нейронів не починатися з нуля.

До речі для множення матриць можна застосовувати Алгоритм Штрассена.
Взагалі множення великих матриць є складним алгоритмом, тому для оптимізації обчислень використовують швидкі мови програмування та обчислення GPU. JavaScript розробник може оптимізувати ці обчислення в браузері користуючись AssemblyScript та WebGL.

Графік сигмоїди



Середньоквадратична похибка (mean squared error, MSE) використовується для *функції втрат* (loss function).
Просто MSE можна записати так: (prediction – actualValue)^2.
Піднесення до степеня необхідне, щоб зробити функцію гладкою, усунути негативні значення, та щоб більші відхилення мали більше значення в сумі.

Після цього виконується *зворотне поширення помилки* (backpropagation) та метод *градієнтного спуску* (gradient descent).
Зворотне поширення обчислює градієнт функції втрат щодо ваг мережі для одного прикладу вхід-вихід і робить це ефективно, обчислюючи градієнт одного шару за раз, ітеруючи назад від останнього шару. Градієнтний спуск або його варіанти, такі як стохастичний градієнтний спуск, зазвичай використовуються.

Кожен шар можна представити як скалярну функцію багатьох змінних.
Ми можемо використати векторний оператор набла (∇) для того, щоб визначити градієнт цієї функції, а по суті набір часткових похідний цієї функції. Коли ми знаємо ці похідні ми можемо оновити ваги певного шару, потім перейти до попереднього шару й також визначити часткові похідні та підібрати параметри так, щоб вони відповідали результатам наступного шару.

Градієнт скалярної функції f записується як:

$\nabla f = (\partial f/\partial x, \partial f/\partial y, \partial f/\partial z)$.

Похідна функція від дійсної змінної

Нехай f — дійсна функція, визначена на відрізку [a, b]. Взявши довільне число $x \in [a, b]$, визначимо:

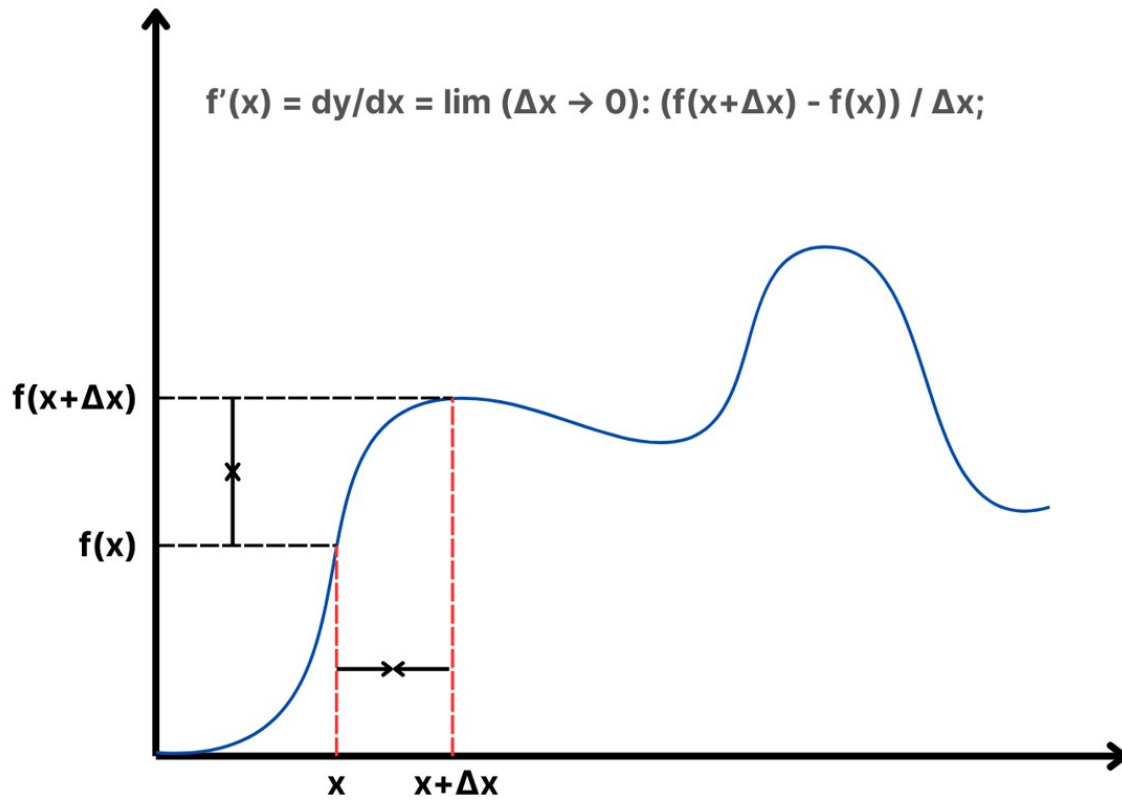
$f'(x) = \lim(t \to x) ((f(t) - f(x)) / (t — x))$, де $a<t<b$, $t \neq x$.

Також:

$f(t) — f(x) = dy$;

$t — x = dx$;

$f'(x) = dy/dx = \lim(t \to x) (dy/dx)$;


f′(x) = dy/dx = lim (Δx → 0): (f(x+Δx) - f(x)) / Δx;
f(x+Δx)
f(x)
x
x+Δx

JavaScript код простої FNN мережі для апроксимації бінарних операторів (XOR, AND)

```
// Функція активації - сигмоїда
function sigmoid(x) {
  return 1 / (1 + Math.exp(-x));
}
// Похідна функції активації
function sigmoidDerivative(x) {
  return x * (1 - x);
}
class NeuralNetwork {
  constructor(inputNodes, hiddenNodes, outputNodes) {
    this.inputNodes = inputNodes;
    this.hiddenNodes = hiddenNodes;
    this.outputNodes = outputNodes;
    // Випадкова ініціалізація ваг
    this.weightsInputHidden = Array(this.hiddenNodes)
      .fill()
      .map(() => Array(this.inputNodes).fill().map(() => Math.random() - 0.5));
    this.weightsHiddenOutput = Array(this.outputNodes)
      .fill()
      .map(() => Array(this.hiddenNodes).fill().map(() => Math.random() - 0.5));
    // Зсуви
    this.biasHidden = Array(this.hiddenNodes).fill().map(() => Math.random() - 0.5);
    this.biasOutput = Array(this.outputNodes).fill().map(() => Math.random() - 0.5);
  }
  // Метод навчання
  train(inputs, targets, learningRate) {
    // --- Feedforward ---
    // Вхідні дані -> Прихований шар
    const hiddenInputs = this.weightsInputHidden.map(weights =>
      weights.reduce((sum, weight, i) => sum + weight * inputs[i], 0)
    );
    const hiddenOutputs = hiddenInputs.map(sigmoid);

    // Прихований шар -> Вихідний шар
    const finalInputs = this.weightsHiddenOutput.map(weights =>
      weights.reduce((sum, weight, i) => sum + weight * hiddenOutputs[i], 0)
    );
    const finalOutputs = finalInputs.map(sigmoid);
    // --- Backpropagation ---
    // Похибки на виході
    const outputErrors = finalOutputs.map((output, i) => targets[i] - output);
    // Похідна функції активації для вихідного шару
    const outputGradients = finalOutputs.map(sigmoidDerivative);
    // Зміна ваг між прихованим і вихідним шарами
    const weightsHiddenOutputDeltas = outputErrors.map((error, i) =>
      hiddenOutputs.map(output => output * error * outputGradients[i] * learningRate)
    );
    this.weightsHiddenOutput = this.weightsHiddenOutput.map((weights, i) =>
      weights.map((weight, j) => weight + weightsHiddenOutputDeltas[i][j])
    );
    // Похибки прихованого шару
    const hiddenErrors = this.weightsHiddenOutput.reduce((errors, weights, i) =>
      errors.map((error, j) => error + weights[j] * outputErrors[i]),
      Array(this.hiddenNodes).fill(0)
```

```
      );
      // Похідна функції активації для прихованого шару
      const hiddenGradients = hiddenOutputs.map(sigmoidDerivative);
      // Зміна ваг між вхідним і прихованим шарами
      const weightsInputHiddenDeltas = hiddenErrors.map((error, i) =>
        inputs.map(input => input * error * hiddenGradients[i] * learningRate)
      );
      this.weightsInputHidden = this.weightsInputHidden.map((weights, i) =>
        weights.map((weight, j) => weight + weightsInputHiddenDeltas[i][j])
      );
    }
    // Метод передбачення
    predict(inputs) {
      // Вхідні дані -> Прихований шар
      const hiddenInputs = this.weightsInputHidden.map(weights =>
        weights.reduce((sum, weight, i) => sum + weight * inputs[i], 0)
      );
      const hiddenOutputs = hiddenInputs.map(sigmoid);
      // Прихований шар -> Вихідний шар
      const finalInputs = this.weightsHiddenOutput.map(weights =>
        weights.reduce((sum, weight, i) => sum + weight * hiddenOutputs[i], 0)
      );
      const finalOutputs = finalInputs.map(sigmoid);

      return finalOutputs;
    }
  }
  // Вхідні дані та очікувані виходи для функції XOR
  const xorInputs = [
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
  ];
  const xorTargets = [
    [0],
    [1],
    [1],
    [0]
  ];
  const inputNodes = 2;
  // Створення мережі
  const nn = new NeuralNetwork(inputNodes, inputNodes + 1, 1);
  // Навчання мережі
  for (let i = 0; i < 100000; i++) {
    const index = Math.floor(Math.random() * xorInputs.length);
    nn.train(xorInputs[index], xorTargets[index], 0.1);
  }
  // Передбачення
  xorInputs.forEach(input => {
    console.log(`Input: ${input} Output: ${nn.predict(input)}`);
  });
```

Навчання, продемонстроване у прикладі XOR, є прикладом навчання з учителем (supervised learning).

JavaScript бібліотеки машинного навчання

За допомогою різних бібліотек машинного навчання ви можете створити програму, яка буде здатна класифікувати дані, та робити передбачення.

Ви можете використовувати JavaScript бібліотеки для роботи з AI в браузері та в Node.js, наприклад:
tensorflow.js,
brain.js,
synaptic.js,
mind.js.

TensorFlow.js може розпізнавати образи з допомогою Згорткових нейронних мереж (CNN). TensorFlow.js визначає тензор як багатовимірний масив, хоча насправді тензор крім своєї структури має спеціальні оператори в тензорному численні, яке розробив італійський фізик-теоретик Тулліо Леві-Чивіта.
TensorFlow.js дозволяє досить просто налаштувати модель, яка буде розпізнавати емоції у вашому тексті, коментарях тощо.

Приклади коду бібліотек ШІ

Brain.js: Приклад “навчання з учителем” для апроксимації оператора XOR

```
// provide optional config object (or undefined). Defaults shown.
const config = {
  binaryThresh: 0.5, // ¯\_(ツ)_/¯
  hiddenLayers: [3], // array of ints for the sizes of the hidden layers in the network
  activation: 'sigmoid' // supported activation types: ['sigmoid', 'relu', 'leaky-relu', 'tanh']
};

// create a simple feed forward neural network with backpropagation
const net = new brain.NeuralNetwork(config);

net.train([{
    input: [0, 0],
    output: [0]
  },
  {
    input: [0, 1],
    output: [1]
  },
  {
    input: [1, 0],
    output: [1]
  },
  {
    input: [1, 1],
    output: [0]
  }
]);

const output = net.run([1, 0]); // [0.987]
```

Mind.js: Оцінка рейтингу фільму

```
const genres = [
    'Action',
    'Adventure',
    'Animation',
    'Comedy',
    'Drama',
    'Family',
    'Musical',
    'Mystery',
    'Romance',
    'Sci-Fi',
    'Sport',
    'Thriller',
    'War'
];

let input = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0];

movie.genres.forEach(function (genre) {
    let index = genres.indexOf(genre);
    if (index > -1) input[index] = 1;
});

// train the network with the rating given by the user, scaled by 5
const mind = Mind()
    .learn([
        { input: input, output: [rating / 5] }
    ]);
```

Tensorflow.js: Знаходження функції (співвідношення) між двома значеннями х та у.

Це навчить нашу нейронну мережу узгоджувати наші х з нашими у, щоб спробувати вивести правила між ними. У нашому випадку у = 2х-1.

```
//<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest">
  async function doTraining(model) {
      const history =
          await model.fit(xs, ys,
              {
                  epochs: 500,
                  callbacks: {
                      onEpochEnd: async (epoch, logs) => {
                          console.log("Epoch:" + epoch
                              + " Loss:" + logs.loss);
                      }
                  }
              });
  }
  const model = tf.sequential();
  model.add(tf.layers.dense({ units: 1, inputShape: [1] }));
  model.compile({
      loss: 'meanSquaredError',
      optimizer: 'sgd'
  });
  model.summary();

  const xs = tf.tensor2d([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], [6, 1]);
  const ys = tf.tensor2d([-3.0, -1.0, 2.0, 3.0, 5.0, 7.0], [6, 1]);

  doTraining(model).then(() => {
      alert(model.predict(tf.tensor2d([10], [1, 1])));
  });
```

Тестування Штучного інтелекту (ШІ)

Класичне тестування зосереджено на помилках у коді, таких як синтаксичні помилки, логічні помилки та помилки виконання. Класичне тестування орієнтоване на функціональність, продуктивність, безпеку та інтерфейс користувача.
Тестування ШІ (Штучного інтелекту) більше зосереджено на точності моделі, якості даних, на яких вона навчалася, її продуктивності на різних наборах даних та на тому, як вона узагальнює нові дані.

Акуратність моделі (точність моделі) — це відношення загальної кількості відповідей, до кількості коректних відповідей. Наприклад, серед 100 відповідей 10 коректних, тоді акуратність буде 10/100;

Перевірка на "перенавчання" (overfitting) та "ненавченість" (underfitting).

Перевірте модель на даних (прикладах), який не було під час навчання.
Перевірте модель на даних, які були під час навчання.

Образно кажучи, перенавчання (overfitting) — це якби двері зробили так, що тільки силует малої групи людей туди може пройти. Тобто занадто сильна підгонка під конкретні якості, які не є універсальними для всієї вибірки.



Або, наприклад, ваша модель може виявляти на картинці вікна тільки квадратної форми, бо ця характеристика часто зустрічалась серед навчальних прикладів.

Для тестування Великих мовних моделей (Large language model, LLM) у сфері обробки природної мови (Natural language processing, NLP) застосовують набір тестів SuperGLUE.
Бенчмарк SuperGLUE (Super General Language Understanding Evaluation) представляє набір складних і різноманітних завдань для оцінки розуміння мови.

Ось декілька основних прикладів задач з набору SuperGLUE:

- BoolQ (Boolean Questions): Задача полягає в відповіді на питання закритого типу (так/ні), яке базується на фрагменті тексту.

- CB (CommitmentBank): Тест на розуміння тривіальних і зобов'язувальних висловлювань у контексті тексту. Модель має визначити, чи стверджує, відкидає, чи залишається нейтральною до тверджень, поданих у тексті.

- COPA (Choice of Plausible Alternatives): Вибір правдоподібної причини або наслідку для даної події з двох варіантів.

- MultiRC (Multi-Sentence Reading Comprehension): Задача з розумінням багатопропозиційних текстів, де необхідно відповісти на питання і вказати, які речення з тексту підтверджують відповідь.

- ReCoRD (Reading Comprehension with Commonsense Reasoning Dataset): Задача, яка вимагає від моделі заповнити пропуски в тексті на основі контексту, використовуючи відповідні імена або замінники.

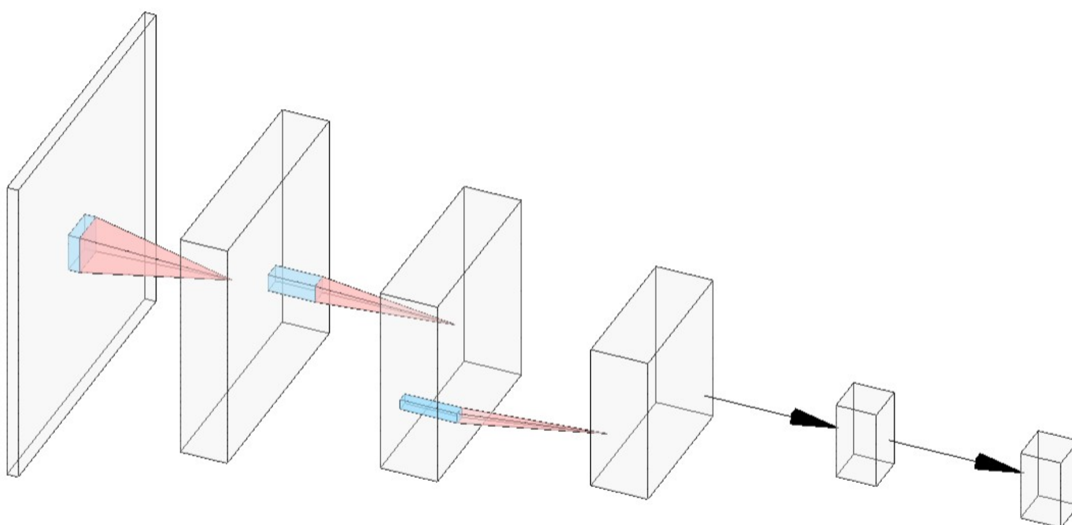
- RTE (Recognizing Textual Entailment): Визначення того, чи логічно випливає одне твердження з іншого.

- WiC (Words in Context): Визначення, чи одне і те ж слово в двох різних реченнях має однакове значення.

Трішки про згорткові нейронні мережі

Згорткові нейронні мережі (Convolutional neural network, CNN) буквально згортаються (convolves)!

Згортова нейронна мережа (CNN) не є повнозв'язною мережею (fully connected network). У CNN зазвичай використовуються шари згортки (convolutional layers) і пулінгу (pooling layers), які спеціалізуються на роботі з просторовими структурами даних, такими як зображення.



Згортковий шар (Convolutional Layer): Він застосовує набір фільтрів, відомих як ядра, до вхідних зображень. Фільтри/ядра — це менші матриці, як правило, у формі 3×3 або 5×5. Ядро (kernel) ковзає по вхідних даних зображення та обчислює скалярний добуток між вагою ядра та відповідним фрагментом вхідного зображення.

Після згортки до результатів застосовується нелінійна функція активації (найчастіше використовується ReLU — Rectified Linear Unit), яка допомагає моделі навчитися нелінійним залежностям.

Рівень об’єднання (Pooling layer): Цей шар зменшує розмір карти ознак, зменшуючи кількість параметрів і обчислювальних ресурсів. Найчастіше використовуються max-pooling або average-pooling.

Шар Вирівнювання (Flattening Layer): Цей шар перетворює багатовимірну карту ознак у

вектор, який може бути подано на вхід до повнозв’язних шарів.

Повнозв’язний Шар (Fully Connected Layer): Це шар, де кожен нейрон з'єднаний з усіма нейронами попереднього шару. Ці шари використовуються для завершення обчислень і виходу кінцевої класифікації чи іншого результату.

Джерела:

“Штучні нейронні мережі: базові положення”, І.А. Терейковський, Д.А. Бушуєв (КПІ),

“Штучні нейронні мережі: обчислення”, М.А. Новотарський, Б.Б. Нестеренко (НАНУ),

"Learning TensorFlow.js: Powerful Machine Learning in JavaScript" by Gant Laborde,

"Deep Learning with JavaScript" by Eric D. Nielsen, Shanqing Cai, and Stanley Bileschi,

"The Hundred-page Machine Learning Book" by Andriy Burkov,

"Deep Learning: A Practitioner's Approach" by Josh Patterson, Adam Gibson,

"Deep learning" by Ian Goodfellow, Yoshua Bengio, Aaron Courville,

“Foundations of Statistical Natural Language Processing” by Christopher D. Manning,

“Pattern Recognition and Machine Learning” by Christopher Bishop,

“Certified Tester AI Testing Syllabus” by ISTQB,

"Attention Is All You Need" Ashish Vaswani, Aidan Gomez,

“Deep Learning Basics: Introduction and Overview” by Lex Fridman (on YouTube),

“MIT Introduction to Deep Learning” by Alexander Amini (on YouTube),

“Lex Fridman - Yann LeCun, Podcast #36” (on YouTube).

https://caza.la/synaptic/#/
https://www.tensorflow.org/js
https://stevenmiller888.github.io/mindjs.net/
https://brain.js.org/#/

https://incrypted.com/ua/shcho-take-nejromerezhi/

https://microsoft.github.io/AI-For-Beginners/
https://stevenmiller888.github.io/mind-how-to-build-a-neural-network/
https://www.ibm.com/topics/deep-learning
https://www.istqb.org/certifications/artificial-inteligence-tester
https://ollama.com/
https://microsoft.github.io/ML-For-Beginners/#/
https://dou.ua/forums/topic/38872/
https://www.ibm.com/topics/embedding
https://www.geeksforgeeks.org/artificial-intelligence-an-introduction/

https://super.gluebenchmark.com/
https://www.mlebook.com/wiki/doku.php
https://en.wikipedia.org/wiki/Attention_Is_All_You_Need
https://developer.nvidia.com/blog/deep-learning-nutshell-core-concepts/
https://archive.org/details/computer-science-ua
https://it-osvita.diia.gov.ua/task/item/2a163d33-3da4-48d2-bbd0-f3a14a7b93da
https://dou.ua/lenta/articles/first-steps-in-nlp-tensorflow/
https://itwiki.dev/data-science/ml-reference/ml-glossary/gradient-boosting
https://www.cloudflare.com/learning/ai/what-are-embeddings/
https://www.turing.com/kb/guide-on-word-embeddings-in-nlp
https://www.w3schools.com/ai/ai_tensorflow_intro.asp
https://medium.com/@jindalkanika02/hello-world-of-tensorflow-js-25650cc12f0e
https://alexlenail.me/NN-SVG/AlexNet.html
https://www.geeksforgeeks.org/tensorflow-js/?ref=lbp
https://www.geeksforgeeks.org/introduction-convolution-neural-network/

https://www.youtube.com/watch?v=Rs_rAxEsAvI
https://www.youtube.com/watch?v=60c4rMq-aH0
https://www.youtube.com/watch?v=RVMHhtTqUxc
https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi